Face Recognition-based Attendance System Technical Report

Image Processing & Computer Vision

CS 3042

Faculty of Computing

General Sir John Kotelawala Defence University

Name: BU Senanayake

Index No: D/BCS/23/0018

1. Introduction

With the advent of Artificial Intelligence and Computer Vision, traditional attendance marking methods have evolved towards automation. This technical report elaborates on the development of a Face Recognition-Based Attendance System built using Python, OpenCV, MTCNN, DeepFace (ArcFace model), ReactJS, Flask, and several supporting libraries. The objective of this system is to improve accuracy and efficiency in student attendance handling and tracking.

2. Objectives

- To develop an attendance marking system using face recognition.
- To implement advanced AI pipelines using Deep Learning models.
- To handle real-time image capture and recognition via socket communication.
- To analyze attendance data and represent it graphically.
- To enable features such as student registration, deletion, and subject-wise attendance tracking.

3. Technologies and Libraries Used

The following technologies were carefully chosen and integrated to fulfill the real-time, AI-based, and user-interactive needs of the system:

Python

Python was used as the main backend programming language due to its rich ecosystem of scientific and AI libraries. It serves as the glue for face detection, embedding generation, file handling, and API hosting.

- Used for: image processing, embedding computations, backend logic, API development
- Why: Python is the industry standard for machine learning and image processing applications

MTCNN (Multi-task Cascaded Convolutional Networks)

MTCNN is used for face detection. It locates and extracts faces from raw webcam images, even in low-light and occluded conditions.

- Used for: detecting faces in captured frames before embedding generation
- Why: High accuracy and reliability in real-world face detection tasks

DeepFace (ArcFace model)

DeepFace is a Python library that wraps state-of-the-art face recognition models. ArcFace was selected for embedding generation due to its strong generalization performance and high accuracy.

- Used for: generating 512-dimensional embeddings from face crops
- Why: ArcFace provides discriminative features that improve matching and classification accuracy

Flask

Flask is a lightweight Python web framework used to build RESTful APIs and manage backend routes for registration, embedding, recognition, and data persistence.

- Used for: /register, /remove-student, /load-embeddings, /process_frame, /save_attendance APIs
- Why: Simple, fast to develop, and supports extensions like Flask-SocketIO for real-time features

OpenCV

OpenCV is a widely used library for computer vision tasks. In this system, it assists in reading, decoding, cropping, and resizing facial images for further processing.

- Used for: decoding base64 images, resizing faces to 112x112, drawing bounding boxes
- Why: Offers highly optimized methods for real-time image manipulation

NumPy

NumPy is the numerical foundation used for handling image matrices, storing and comparing embeddings efficiently.

- Used for: converting, normalizing, storing embeddings, calculating cosine similarity
- Why: High-speed operations on numerical data; supports .npy format for quick read/write

ReactJS

React was used for the frontend interface to capture user inputs, webcam images, and interact with the backend.

- Used for: interfaces like RegisterPage, CapturePage, TakeAttendenceCamera, and data visualizations
- Why: Enables fast development and component-based UI logic

Socket.IO (Flask-SocketIO)

Socket.IO handles real-time communication between frontend and backend. It streams webcam frames to the server and receives processed face recognition events.

- Used for: sending images frame-by-frame to the server for recognition
- Why: Reduces latency and allows continuous face detection without refreshing the page

Pandas

Pandas is used for data manipulation and storage. It allows easy creation, editing, and analysis of Excel files that store attendance records.

- Used for: reading and writing .xlsx files, appending attendance entries, removing students
- Why: Provides Excel-like operations in memory, which are easy to manipulate programmatically

4. System Architecture Overview and User Interfaces

The system is composed of:

- A ReactJS-based frontend for user interaction.
- A Flask backend with REST and WebSocket endpoints.
- A face embedding pipeline powered by ArcFace and MTCNN.
- Excel sheet management using Pandas.
- Real-time attendance marking through webcam stream.

AI Pipelines and Features

1. **Face Detection (MTCNN)**
   Detects faces with high accuracy even in challenging conditions like side profiles, shadows, or partial occlusions.
2. **Face Embedding Generation (ArcFace via DeepFace)**
   Converts faces into 512-dimensional feature vectors for robust identity representation using state-of-the-art ArcFace model.
3. **Cosine Similarity Matching**
   Compares real-time face embeddings with stored ones using cosine similarity — a widely accepted metric in facial recognition.
4. **Embedding Normalization**
   Normalized vectors improve recognition consistency and reduce sensitivity to lighting or scale variations.
5. **Mean Embedding from 30 Samples**
   Combines 30 captured face images to create one reliable average embedding, improving accuracy.
6. **Color-coded Bounding Boxes**
   Bounding box color changes (green, yellow, red) based on match confidence, improving user understanding immediately.
7. **Dynamic Embedding Loading**
   Loads only the embeddings for selected intakes and courses, improving memory efficiency and recognition speed

5. Summary of Backend Components

5.1 Student Registration

5.1.1 Capturing 30 Photos

The registration flow begins with capturing 30 face images of the student through their webcam using CapturePage.jsx. We take 30 photos to capture different angles and lighting conditions to create a more accurate and reliable face embedding. The delay between frames ensures variation in expressions and angles.

This ensures diversity in facial samples which helps in creating robust embeddings.

```
const images = []
setIsCapturing(true)
setCaptureProgress(0)

for (let i = 0; i < 30; i++) {
  context.drawImage(video, 0, 0, canvas.width, canvas.height)
  const imageData = canvas.toDataURL('image/jpeg')
  images.push(imageData)

  setCaptureProgress(Math.round(((i + 1) / 30) * 100))
  // to wait between frames to get diffrent actions and lighting confitions of the face
  await new Promise((resolve) => setTimeout(resolve, 150))
}
```

5.1.2 Creating Embeddings from Captured Images

Once captured, the 30 images are sent to the Flask backend via the /register API. The function create_student_embeddings() processes each image:

- Decode base64 to image.
- Detect face using MTCNN.
- Crop the face using coordinates.
- Resize to 112x112 pixels.
- Pass to ArcFace via DeepFace for embedding extraction
- Take an average of 30 embeddings for final.

```
    # Detect face
    results = detector.detect_faces(img)
    if not results:
        print(f"No face detected in image {i}")
        continue

    x, y, w, h = results[0]['box']
    x, y = max(0, x), max(0, y)
    cropped = img[y:y + h, x:x + w]

    if cropped.size == 0:
        print(f"Cropped face in image {i} is empty")
        continue

    # Resize and preprocess
    cropped = cv2.resize(cropped, dsize: (112, 112))

    # Get embedding
    try:
        emb = DeepFace.represent(cropped, model_name='ArcFace', enforce_detection=False)[0]['embedding']
        embeddings.append(emb)
        valid_images += 1
        print(f"Processed image {i} successfully")
    except Exception as e:
        print(f"Embedding error for image {i}: {e}")

except Exception as e:
    print(f"Error processing image {i}: {e}")
```

This function uses **detector.detect_faces(img)** to detect a face, **img[y:y + h, x:x + w]** to crop the face, **cv2.resize()** to resize it to 112×112 pixels, and **DeepFace.represent(..., model_name='ArcFace')** to generate a face embedding, which is then stored in the **embeddings** list if successful.

5.1.3 Storing Embeddings Efficiently

All embeddings are stored in the format:
Students/{intake}/{course}/{name}_{studentid}.npy

Why NumPy?

- Fast disk read/write
- Optimized for computing
- Easily batch-loaded into memory

```python
# Save mean embedding
if embeddings:
    mean_embedding = np.mean(embeddings, axis=0)
    filename = f"{name}_{studentid}.npy"
    np.save(os.path.join(folder_path, filename), mean_embedding)
    print(f"Created embedding from {valid_images} valid images")
    return {"success": f"Embedding created from {valid_images} images"}
else:
    print("No valid faces found in the images")
    return {"error": "No valid faces found in the images"}
```

This code checks if any embeddings were created, then uses **np.mean(embeddings, axis=0)** to compute the average embedding, saves it as a .npy file using **np.save()**, and returns a success message indicating how many valid images were used.

5.2 Student Face Detection and Recognition

5.2.1 Loading and Normalizing Embedding

Before real-time detection, embeddings for selected intakes and courses are loaded into memory

```python
    embedding = np.load(os.path.join(path, file))
    embedding = embedding / np.linalg.norm(embedding)
    loaded_embeddings[name] = embedding
```

This allows fast cosine similarity comparison

```python
# Compute similarities in one go
emb_normalized = emb / np.linalg.norm(emb)
similarities = np.dot(embeddings_matrix, emb_normalized)

# Find best match
best_idx = np.argmax(similarities)
best_similarity = similarities[best_idx]
best_match = names[best_idx]
```

Normalizing the new face embedding using emb / np.linalg.norm(emb) so it has unit length for accurate cosine similarity comparison.

Computing similarities between the new face and all stored embeddings at once using np.dot(embeddings_matrix, emb_normalized), which performs a fast dot product to get similarity scores.

```
for intake_item in intakes:
    for course_item in courses:
        intake = intake_item[0] if isinstance(intake_item, list) else intake_item
        course = course_item[0] if isinstance(course_item, list) else course_item

        path = os.path.join(base_path, intake, course)
        if os.path.exists(path):
            for file in os.listdir(path):
                if file.endswith('.npy'):
                    name = file.replace( __old: '.npy', __new: '')
                    try:
                        embedding = np.load(os.path.join(path, file))
                        embedding = embedding / np.linalg.norm(embedding)
                        loaded_embeddings[name] = embedding
                        count += 1
                    except Exception as e:
                        print(f"Error loading embedding {file}: {e}")

print(f"Loaded {count} embeddings")
return count
```

- Loops through each intake_item and course_item
- Builds the path using os.path.join(base_path, intake, course)
- Checks if the folder exists with os.path.exists(path)
- Finds all .npy files using file.endswith('.npy')
- Loads embeddings using np.load()
- Normalizes them using embedding / np.linalg.norm(embedding)
- Store them in loaded_embeddings with the student's name as key
- Counts how many embeddings were successfully loaded with count += 1

5.2.2 Real-Time Recognition Pipeline

Images streamed from webcam are handled by /process_frame socket handler. Steps include:

- Detecting face using MTCNN
- Cropping and resizing
- Generating embedding
- Compared with loaded .npy files

```
# Detect faces in the image
faces = detector.detect_faces(img_np)
if not faces:
    print("⚠️ No face detected")
    return jsonify({"message": "No face detected"}), 400

# Process the detected face
face = faces[0]
x, y, w, h = face["box"]
print(f"🔍 Face detected at: {x},{y},{w},{h}")

# Crop and preprocess the face
cropped_face = img_np[y:y + h, x:x + w]
cropped_face = cv2.resize(cropped_face,  dsize: (112, 112))

# Generate embedding for the detected face
try:
    new_embedding = DeepFace.represent(cropped_face, model_name='ArcFace', enforce_detection=False)[0][
        'embedding']
except Exception as e:
    print(f"❌ Embedding error: {e}")
    return jsonify({"message": "Failed to generate face embedding"}), 400
```

This detects a face in the image using detector.detect_faces, crops and resizes it with cv2.resize, then generates a face embedding using DeepFace.represent with the ArcFace model. If no face is detected or embedding fails, it returns an error message in terminal.

5.2.3 Matching and Thresholds for Recognition

```
# Find the best match
best_match = None
best_similarity = 0

for name, saved_emb in loaded_embeddings.items():
    similarity = cosine_similarity(new_embedding, saved_emb)
    print(f"Similarity with {name}: {similarity:.4f}")

    if similarity > best_similarity:
        best_similarity = similarity
        best_match = name
```

Above code compares the new face embedding with each saved embedding using cosine_similarity, prints the similarity score, and keeps track of the highest similarity and corresponding name to identify the best match.

```
threshold = 0.75
if w < 30:  # Far faces
    threshold = 0.72



if best_similarity <= threshold:
    print(
        f"⚠️ Face #{i + 1} - Best match below threshold: {best_match} ({best_similarity:.4f} < {threshold})")
    socketio.emit( event: 'below_threshold_match', *args: {
        'name': best_match,
        'similarity': float(best_similarity),
        'threshold': threshold,
        'box': [x, y, w, h]
    })
    continue

# Set a stricter threshold for more accurate recognition
if best_similarity > threshold:
    # Check if we just recognized this person to avoid duplicates
    current_time = time.time()
    if best_match in recent_recognitions:
        # Only re-emit if it's been at least 1 second
        if current_time - recent_recognitions[best_match] < 1.0:
            continue
```

This code sets a similarity threshold (default 0.75) to decide whether a face match is reliable. If the face is small (likely farther away), the threshold is slightly lowered to 0.72 for flexibility. It then compares the best similarity score:

If it's below or equal to the threshold, the match is rejected, and a warning is emitted using socketio.emit('below_threshold_match').

If it's above the threshold, the match is accepted, but it checks if the person was just recognized recently (within 1 second) to avoid duplicate recognition events.

This logic helps balance accuracy and responsiveness in real-time face recognition.

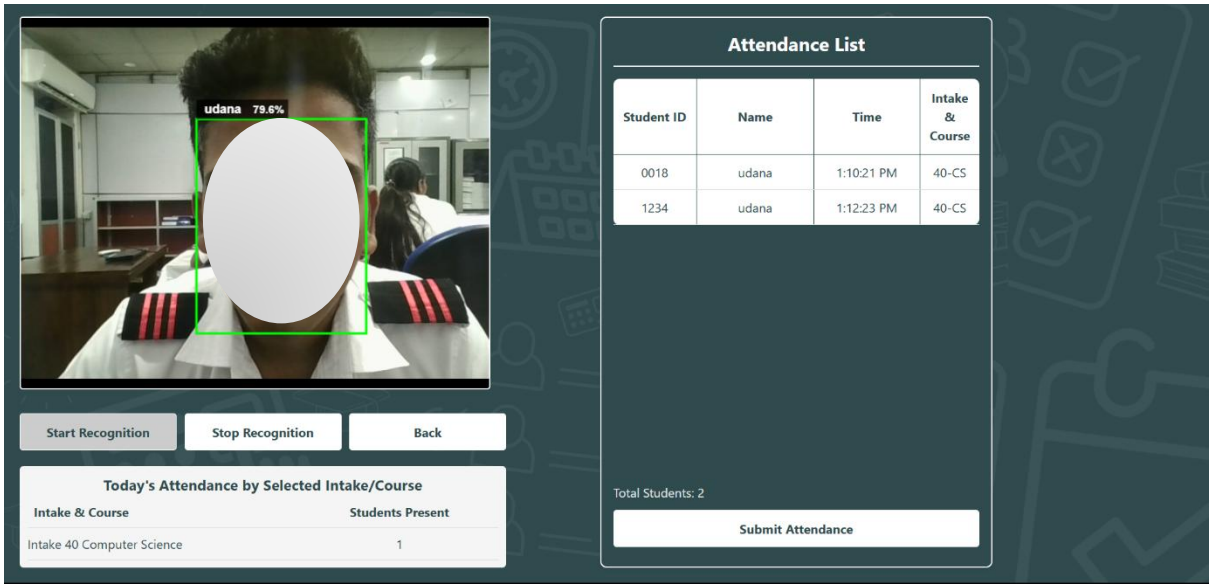
```
    // Draw face box
    ctx.strokeStyle = color;
    ctx.lineWidth = 3;
    ctx.strokeRect(boxX, boxY, boxWidth, boxHeight);

    ctx.font = 'bold 16px Arial';
    const nameText = face.name;
    const textWidth = ctx.measureText(nameText).width;
    const labelY = Math.max(25, boxY);

    ctx.fillStyle = '□rgba(0, 0, 0, 0.7)';
    ctx.fillRect(boxX, labelY - 25, textWidth + 20, 25);
    ctx.fillStyle = '#FFFFFF';
    ctx.fillText(nameText, boxX + 10, labelY - 7);


    if (face.similarity) {
        const similarityText = `${(face.similarity * 100).toFixed(1)}%`;
        ctx.font = 'bold 14px Arial';
        const simWidth = ctx.measureText(similarityText).width;
        ctx.fillStyle = '□rgba(0, 0, 0, 0.7)';
        ctx.fillRect(boxX + textWidth + 20, labelY - 25, simWidth + 10, 25);
        ctx.fillStyle = '#FFFFFF';
        ctx.fillText(similarityText, boxX + textWidth + 25, labelY - 7);
    }
:
// Draw recognized faces (green)
activeFaces.forEach(([id, face]) => {
    drawFaceBox(face, '#00FF00'); // Green for recognized
});

// Draw unrecognized faces (red)
activeUnrecognizedFaces.forEach(([id, face]) => {
    drawFaceBox(face, '#FF0000'); // Red for unrecognized
});

// Draw below threshold faces (yellow)
activeBelowThresholdFaces.forEach(([id, face]) => {
    drawFaceBox(face, '#FFCC00'); // Yellow for below threshold
});
```

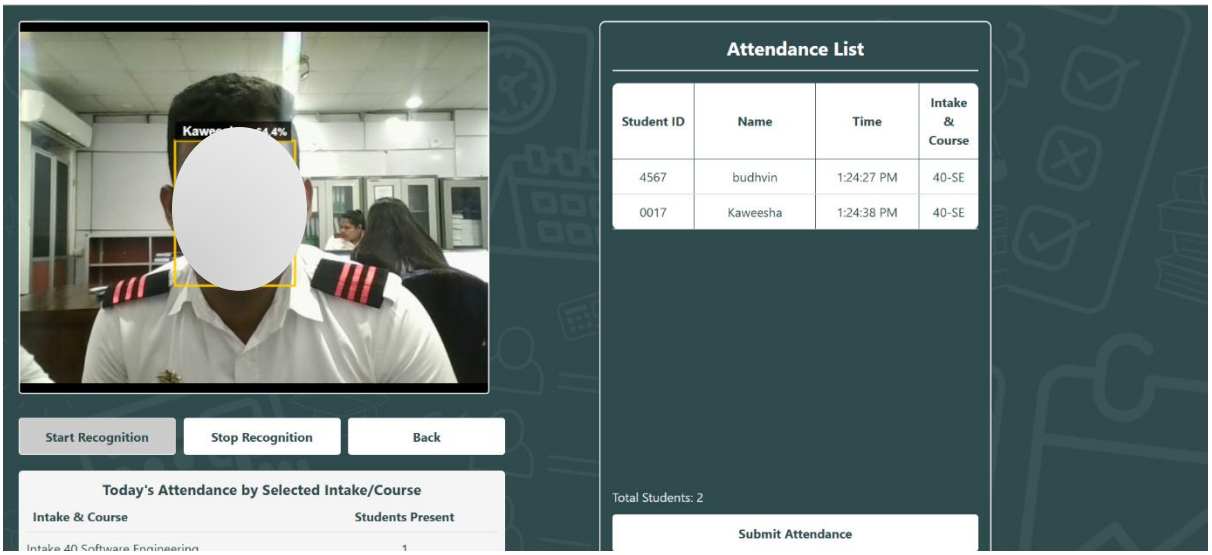
A threshold value (usually 0.75) is used in the backend to decide whether a detected face is confidently recognized. This result is then sent to the frontend, where the bounding box is drawn in different colors based on recognition status:

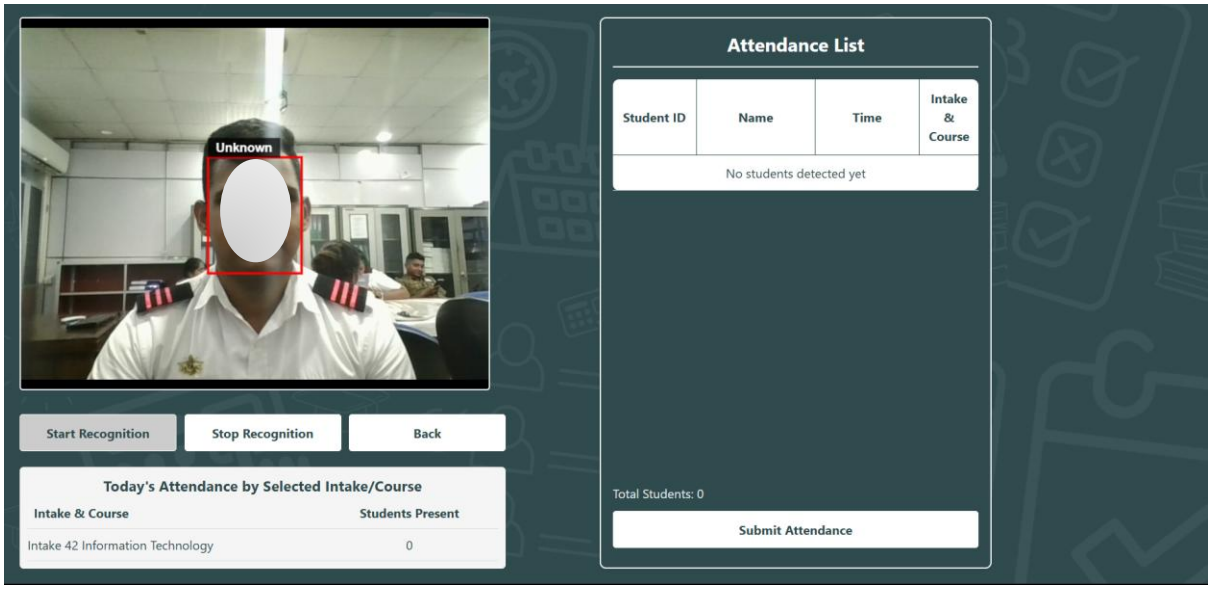
- 🟩 Green: if similarity is greater than the threshold (face is recognized)
- 🟧 Yellow: if similarity is below or equal to the threshold (match found but not confident)
- 🟥 Red: if no match is found (face not recognized)

These conditions determine the color of the bounding box drawn on the canvas in the front end.

Similarity (%)	Meaning	Color
>=75%	Recognized face	Green
60–74%	Low confidence	Yellow
< 60%	Not recognized	Red




Start Recognition
Stop Recognition
Back
Today's Attendance by Selected Intake/Course
Intake & Course
Students Present
Attendance List
Student ID
Name
Time
Intake & Course
4567
budhvin
1:24:27 PM
40-SE
0017
Kaweesha
1:24:38 PM
40-SE
Total Students: 2
Submit Attendance


Unknown
Start Recognition
Stop Recognition
Back
Today's Attendance by Selected Intake/Course
Intake & Course
Students Present
Intake 42 Information Technology
0
Attendance List
Student ID
Name
Time
Intake & Course
No students detected yet
Total Students: 0
Submit Attendance

5.3 Attendance Recording and Excel Management

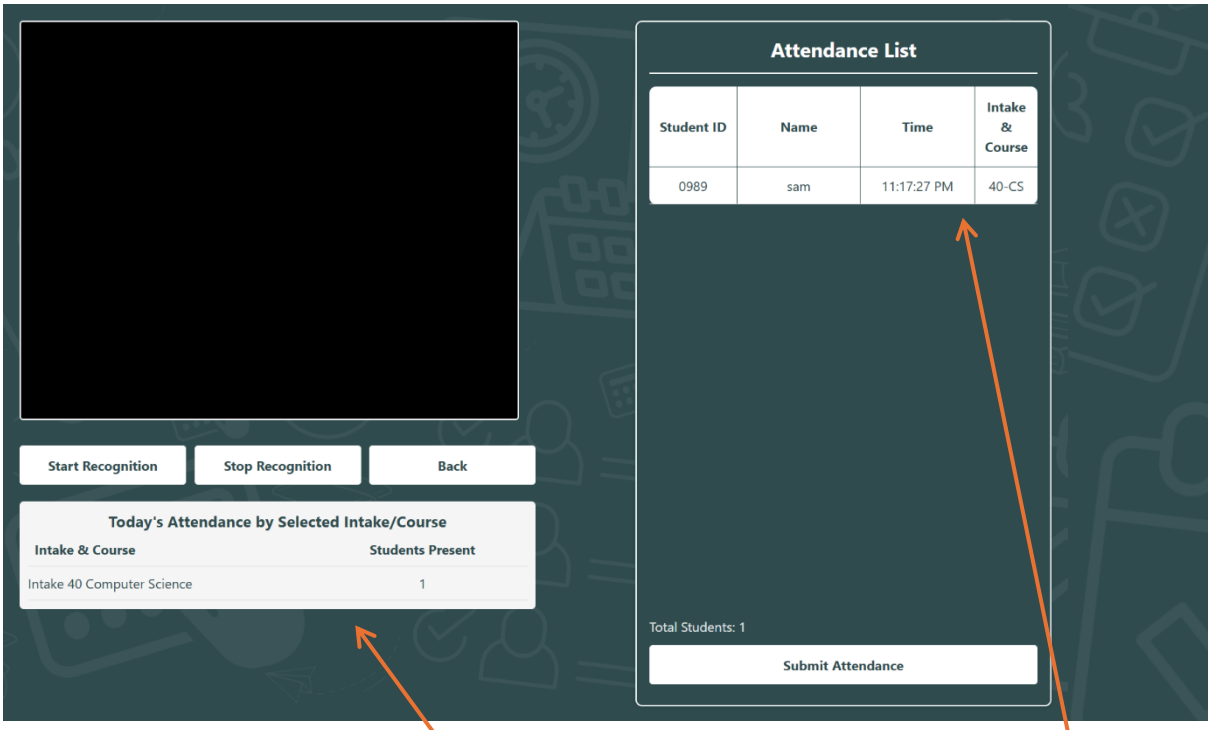
5.3.1 Attendance Table and Attendance Summery

```
setAttendanceList(prevList => {
    if (!prevList.some(s => s.id === studentId)) {
        updateAttendanceSummary(studentIntakeCourse);


        const newStudentEntry = {
            id: studentId,
            name: studentName,
            time: currentTime,
            intakeCourse: studentIntakeCourse,
            formattedIntakeCourse: formattedIntakeCourse
        };

        return [...prevList, newStudentEntry];
    }
    return prevList;
```

Above code checks if the student ID already exists in the attendanceList. If not, it updates the daily summary using updateAttendanceSummary(), creates a newStudentEntry with student details (ID, name, time, intake, course), and adds it to the list using setAttendanceList. If the student is already listed, it just returns the existing list.

```
const updateAttendanceSummary = (intakeCourse) => {
    console.log(`Updating attendance summary for ${intakeCourse}`);

    const currentCount = attendanceList.filter(
        student => student.intakeCourse === intakeCourse
    ).length;


    setAttendanceSummary(prev => {
        const newSummary = {...prev};
        newSummary[intakeCourse] = currentCount + 1;
        console.log(`Set count for ${intakeCourse} to ${newSummary[intakeCourse]}`);
        return newSummary;
    });
};
```

This function updateAttendanceSummary() updates the count of present students for a specific intakeCourse. It first counts how many students in the current attendanceList belongs to that intake/course, adds 1 for the newly detected student, and then updates the attendanceSummary state using setAttendanceSummary. This keeps track of how many students are recognized for each intake and course during the session.



5.3.2 Excel Management

```
new_row = pd.DataFrame([{
    "Name": name,
    "StudentID": studentid,
    "Intake": intake,
    "Course": course,
    "Date": today
}])

if os.path.exists(file_path):
    df = pd.read_excel(file_path)

    # 🔒 Convert all StudentID values to string and strip spaces
    df['StudentID'] = df['StudentID'].astype(str).str.strip()
    cleaned_studentid = str(studentid).strip()

    # ✅ Now check for duplicates
    if cleaned_studentid in df['StudentID'].values:
        print(f"Duplicate found for ID: {cleaned_studentid}")
        return jsonify({"message": "Student ID already registered!"}), 400

    df = pd.concat( objs: [df, new_row], ignore_index=True)

else:
    df = new_row

df.to_excel(file_path, index=False)
print(f"Registered: {studentid} - {name}")
return jsonify({"message": "Student registered successfully!"})
```

This code handles student registration and saves the details in an Excel file using the pandas (pd) library. It first creates a new row with student info (name, ID, intake, course, date). Then it checks if the Excel file exists using the os module. If it does, it loads the data with pd.read_excel(), converts all IDs to strings, checks for duplicate Student IDs, and if not found, adds the new row using pd.concat(). If the file doesn't exist, it creates a new DataFrame. Finally, it saves the updated data back to Excel with df.to_excel().

	A	B	C	D	E	F
1	**Name**	**StudentID**	**Intake**	**Course**	**Date**	
2	sam	989	Intake 40	Computer Science	2025-04-06	
3	udana	1234	Intake 40	Computer Science	2025-04-07	

```python
if os.path.exists(file_path):
    df = pd.read_excel(file_path)
else:
    df = pd.DataFrame(columns=["StudentID", "Name"])

# Ensure 'StudentID' is string
df['StudentID'] = df['StudentID'].astype(str)

# Mark attendance
for index, row in df.iterrows():
    student_id = str(row['StudentID'])
    df.at[index, today] = '✅' if student_id in present_ids else '❌'

# Add new students if not already in the sheet
for student_id, name in present_ids.items():
    if not (df['StudentID'] == student_id).any():
        new_row = {
            "StudentID": student_id,
            "Name": name,
            today: '✅'
        }
        df = pd.concat( objs: [df, pd.DataFrame([new_row])], ignore_index=True)

df.to_excel(file_path, index=False)
return jsonify({"message": "Attendance saved successfully!"})
```

This code saves the attendance of students into an Excel file using the pandas library. It first checks if the file exists using os.path.exists(). If it does, it reads the file with pd.read_excel(), otherwise it creates a new DataFrame with "StudentID" and "Name" columns. It ensures all Student IDs are treated as strings. Then, it marks today's attendance with ✅ if the student is present or ❌ if not. Finally, it adds any new students not already in the sheet with their attendance marked as ✅.

	A	B	C	D	E	F	G	H	I
1	StudentID	Name	14/2/2025	3/3/2025	10/3/2025	17/3/2025	24/3/2025	1/4/2025	7/4/2025
2	1201	sam	☑	☑	✗	✗	☑	✗	☑
3	1202	john	☑	☑	☑	☑	☑	☑	☑
4	1203	amal	✗	☑	☑	☑	☑	☑	☑
5	1204	nimal	☑	✗	☑	☑	☑	☑	✗
6	1205	kamal	☑	☑	☑	✗	☑	☑	☑
7	1206	bimal	☑	☑	☑	☑	☑	✗	☑
8	1207	namal	☑	✗	✗	☑	☑	☑	☑
9	1208	kusumal	✗	☑	☑	☑	✗	☑	☑
10	1209	gayani	☑	☑	☑	☑	☑	☑	☑
11	1210	piyumi	☑	✗	☑	☑	☑	☑	✗
12	1211	haritha	☑	☑	☑	✗	☑	✗	✗
13	1212	thushara	☑	☑	☑	☑	☑	✗	☑
14	1213	ruvi	✗	☑	✗	☑	✗	☑	☑

5.4 Deleting a Student and Their Embedding

5.4.1 Removing Student

```
# Normalize for case-insensitive comparison
df['StudentID'] = df['StudentID'].astype(str).str.strip()
df['Name'] = df['Name'].astype(str).str.strip().str.lower()

# Check if row exists
match = (df['StudentID'] == studentid) & (df['Name'] == name)

if match.sum() == 0:
    return jsonify({"message": "No matching student found."}), 404

# Remove matching rows
df = df[~match]

# Save the updated file
df.to_excel(file_path, index=False)
```

This part of the code uses the pandas (pd) library to manage Excel data. It first reads the Excel file using pd.read_excel(), then normalizes the columns (like converting StudentID and Name to lowercase and stripping spaces) to ensure accurate matching. Using a condition, it finds the row where both the name and student ID match the given input, removes that row from the DataFrame using df = df[~match], and saves the updated file using df.to_excel(). This ensures the student's record is removed from the attendance file.

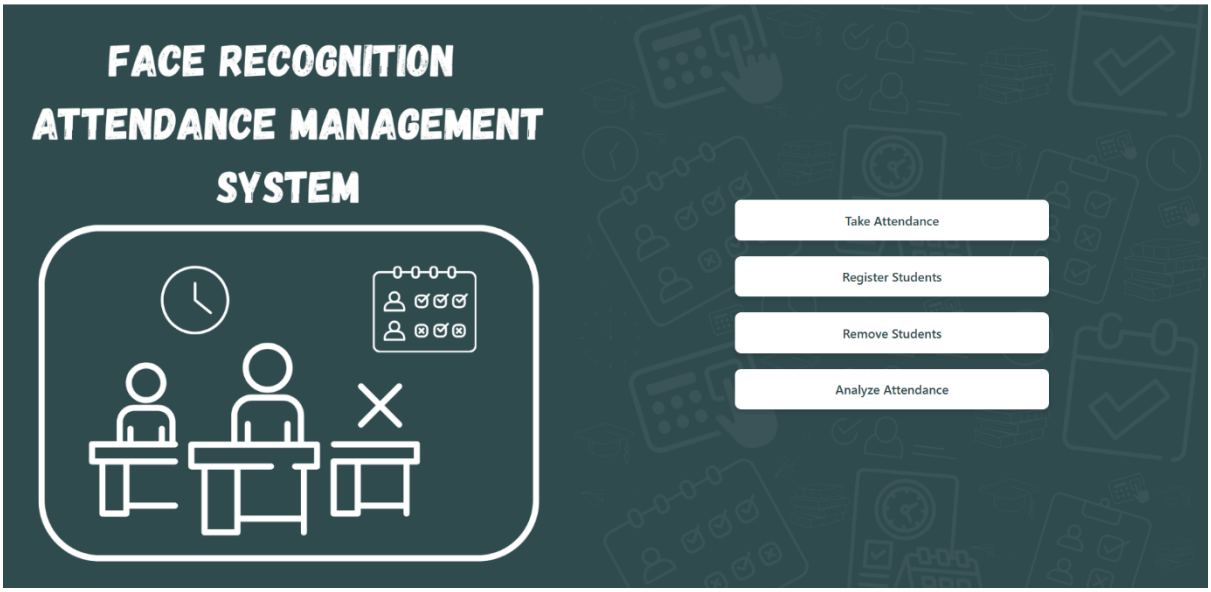
5.4.1 Removing Student

```
embedding_path = os.path.join(f"Students/{intake}/{course}", f"{name}_{studentid}.npy")
if os.path.exists(embedding_path):
    os.remove(embedding_path)
    print(f"Removed embedding file: {embedding_path}")

return jsonify({"message": "Student removed successfully!"})
```

This part uses the os library to handle files. It builds the file path for the .npy embedding using os.path.join(), then checks if the file exists using os.path.exists(). If the embedding file is found, it deletes it using os.remove(). This completely removes the student's face data from the system, ensuring no future recognition can occur using that embedding.

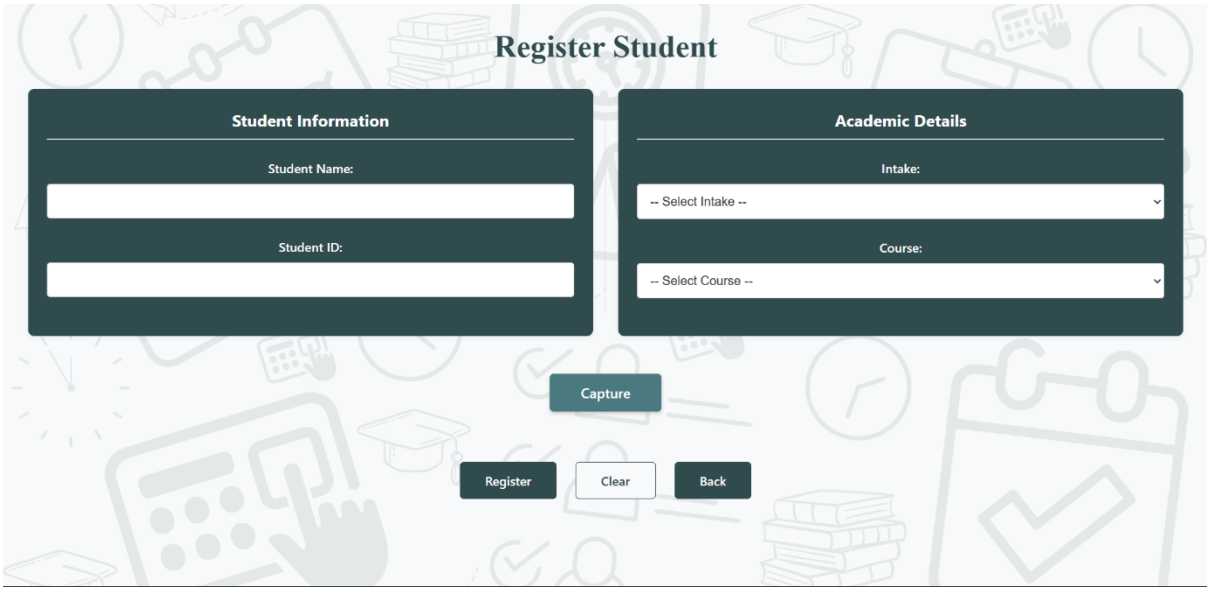
6. Summary of ReactJS Components

6.1 Main Page

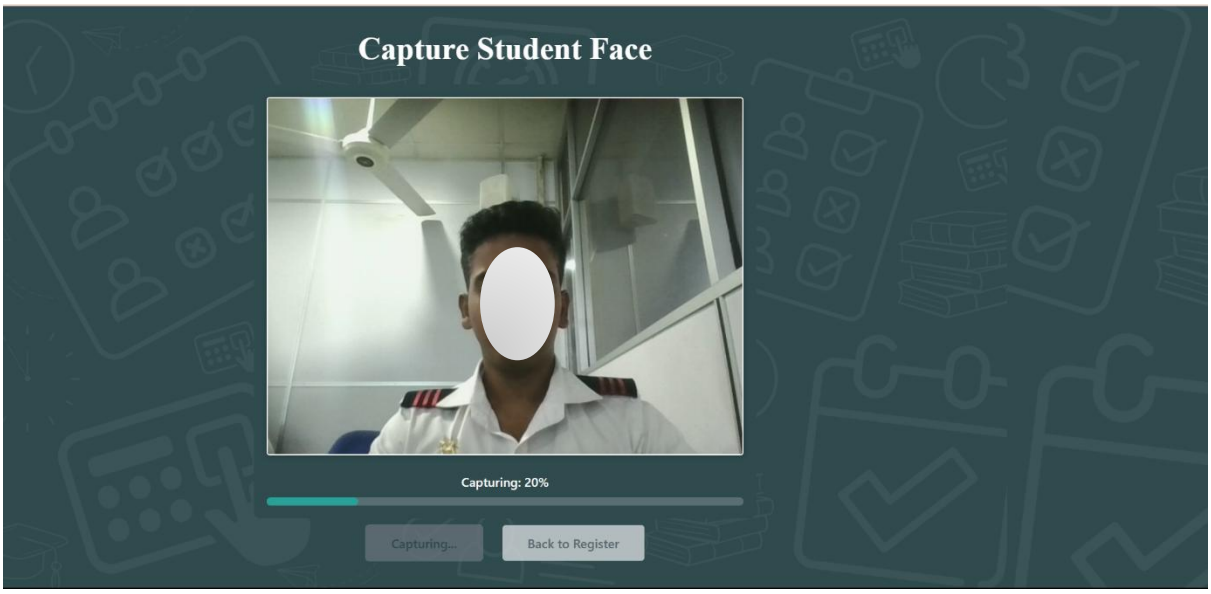


The MainPage.jsx is the homepage of the attendance system. It shows four buttons: Register Students, Take Attendance, Remove Students, and Analyze Attendance. When a button is clicked, it takes the user to the related page using React Router. The page has a background image on the left and buttons on the right. It uses React and some CSS for layout and style.

6.1 Student Registration



This RegisterPage.jsx file is the React component for registering a student. It uses a form to collect the student's name, student ID, intake, and course, and also allows capturing 30 face images for recognition. When the Register button is clicked, it checks if 30 photos are available, then sends all the data to the Flask backend at http://localhost:5000/register for saving in an Excel file and creating embeddings. It uses localStorage to temporarily save form data and photos. Buttons are provided to Capture photos, Clear the form, and Go Back to the main page.

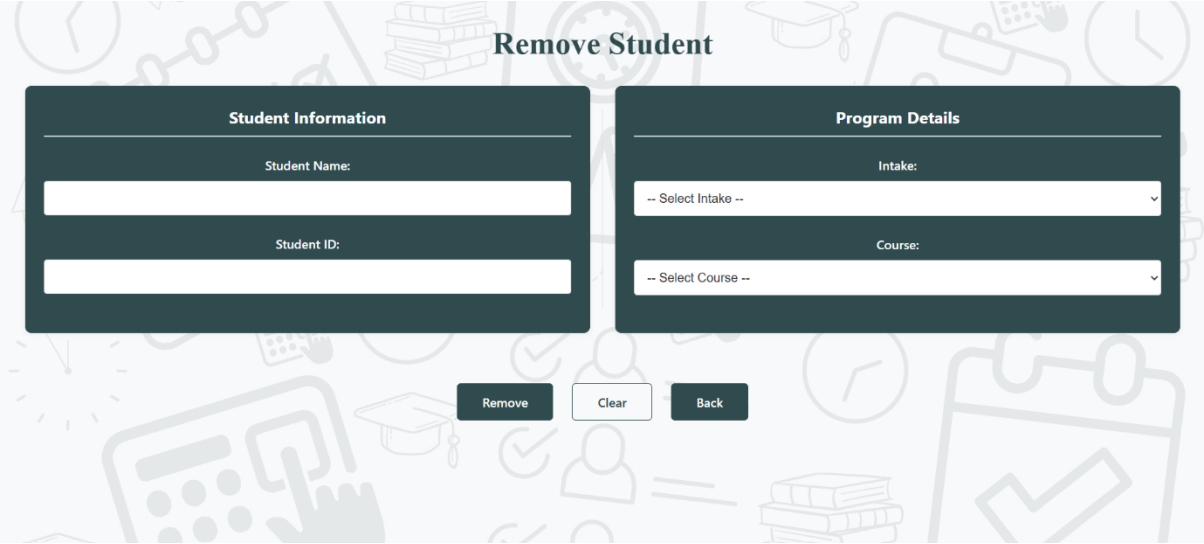


This CapturePage.jsx React component captures 30 photos from the webcam to register a student's face. It uses a <video> element to show the camera feed and a hidden <canvas> to capture images. When the "Capture Photos" button is clicked, it takes 30 snapshots with short delays between each one to capture different expressions and lighting conditions. These images are stored in localStorage as "capturedPhotos" to be used later in the registration form. It shows a progress bar while capturing, and after completion, it navigates back to the RegisterPage.

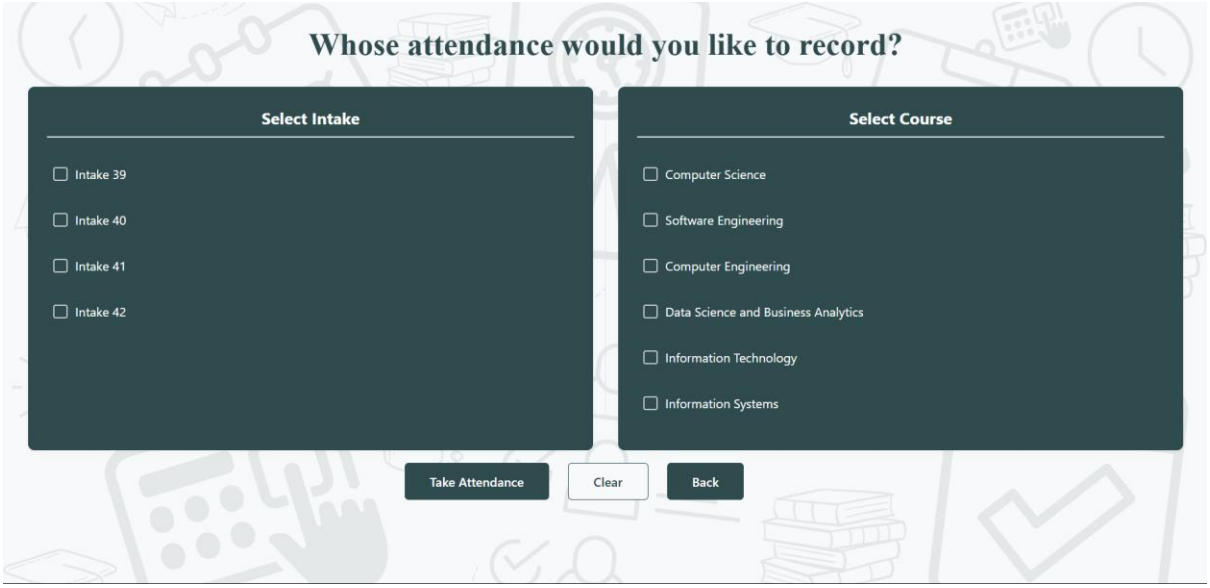
6.2 Student Deregistration

This RemoveStudentPage.jsx code creates a simple React form that allows users to remove a student from the system by entering their name, student ID, intake, and course. When the Remove button is clicked, the form data is sent via a POST request to the backend endpoint http://localhost:5000/remove-student. If the backend finds a match, it deletes the student's record from the Excel file and also deletes their embedding file used for face recognition.

The code also includes a Clear button to reset the form and a Back button to navigate to the home page. Key technologies include React hooks (useState, useNavigate) and the Fetch API

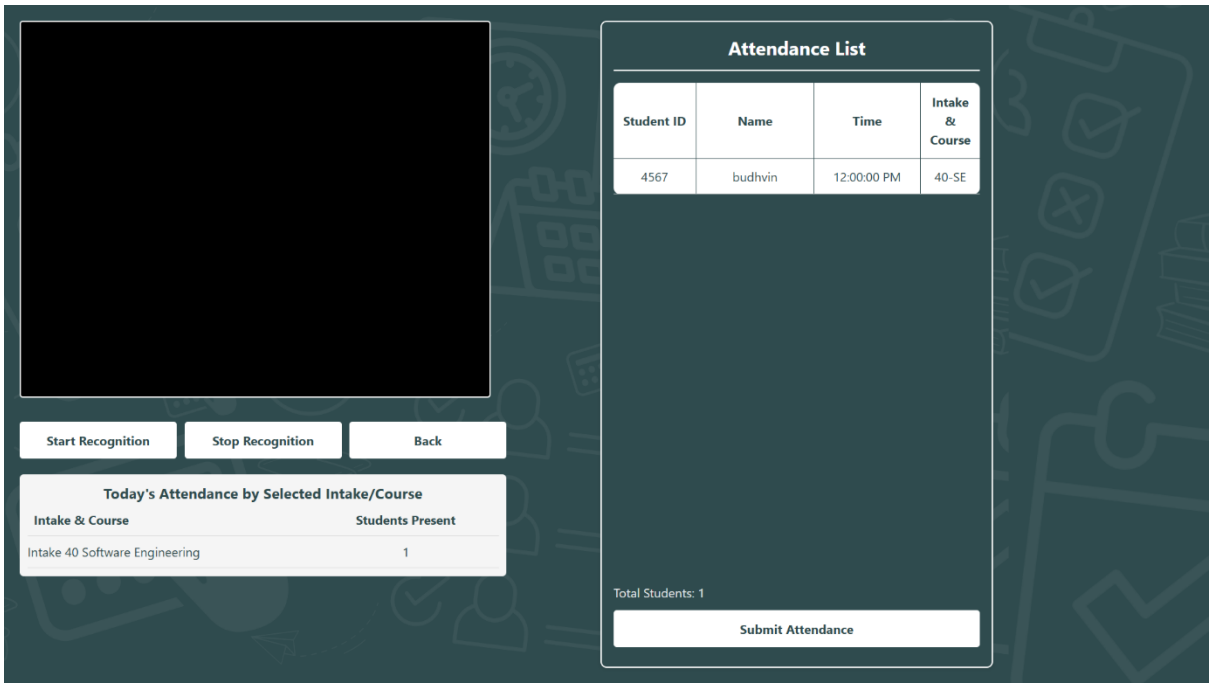


6.3 Taking Attendance



This AttendencePage.jsx code creates a simple user interface that allows the user to select intakes and courses for which they want to record attendance. The user selects options using checkboxes. When the "Take Attendance" button is clicked, the selected data is sent to the backend (/load-embeddings API), which loads the corresponding face embeddings needed for recognition.

If the loading is successful, it navigates to the camera page (/takeattendencecamera) where face recognition begins. The form also includes "Clear" to reset selections and "Back" to return to the main page. Key technologies used include React hooks, fetch API, React Router, and form handling.

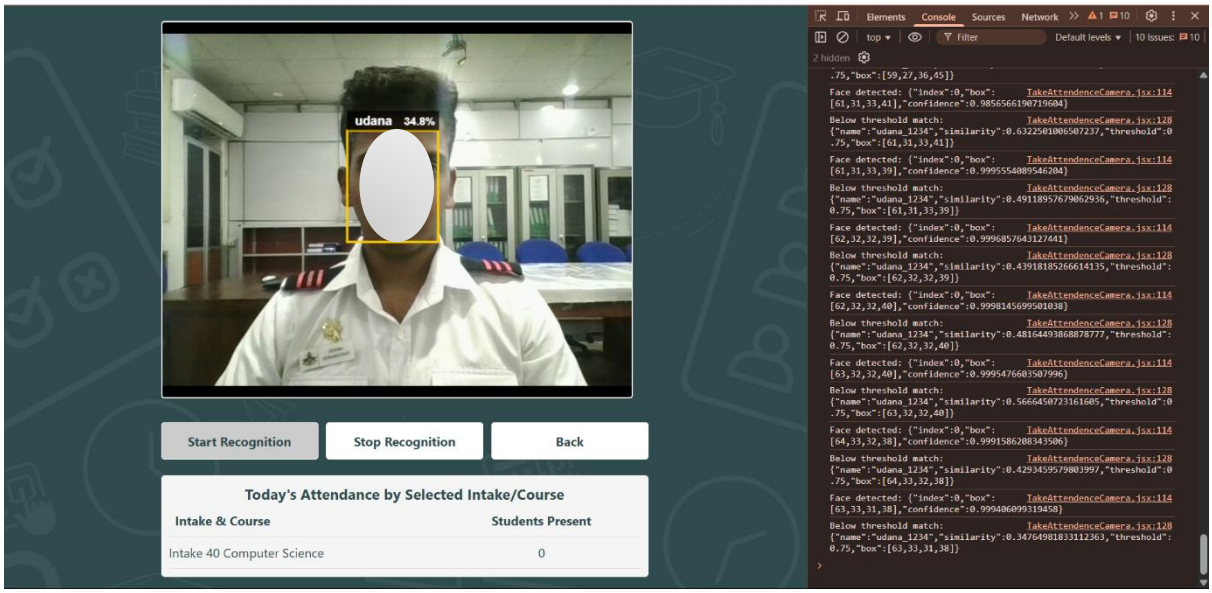


This TakeAttendenceCamera.jsx file handles the **real-time face recognition and attendance tracking** part of your application using React and Socket.IO.

1. **Webcam Access**:
   When the user clicks “Start Recognition,” the app accesses the webcam and starts streaming video.
2. **Send Frames to Backend**:
   It captures frames continuously every 50ms and sends them to the backend for face recognition.
3. **Receive Recognition Results**:
   - If the backend detects and recognizes a face with **similarity ≥ 0.75**, it adds the student’s name, ID, time, and intake/course to the attendance list.
   - If the face is **below threshold**, it is marked yellow.
   - If **unrecognized**, it is marked in red.

4. **Draw Bounding Boxes**:
    - Green: recognized face (above threshold).
    - Yellow: below-threshold match.
    - Red: unknown face. These are drawn on a <canvas> over the video using coordinates received from the backend.
5. **Attendance List & Summary**:
   The right panel shows each student detected with name, ID, time, and intake/course. The left panel shows a summary table of how many students were recognized from each selected intake and course.
6. **Submit & Navigation**:
    - “Submit Attendance” navigates to a confirmation page with recognized students.
    - “Back” stops recognition and returns to the intake/course selection.

This component ties together the webcam, face detection, similarity thresholding, and attendance list updates, all in real-time.



These console logs show that the system is detecting faces through the webcam and trying to recognize them. Each time a face is detected, the system logs its position and confidence level. Then, it compares the face with registered students. If the similarity between the detected face and a stored student face is lower than the set threshold (like 0.72 or 0.75), it logs a "below threshold match."

This means the system found a possible match (e.g., “udana_1234” or “sam_0989”), but the similarity score wasn’t high enough to confidently confirm the identity. So, the face is not marked as recognized and won’t be counted for attendance.

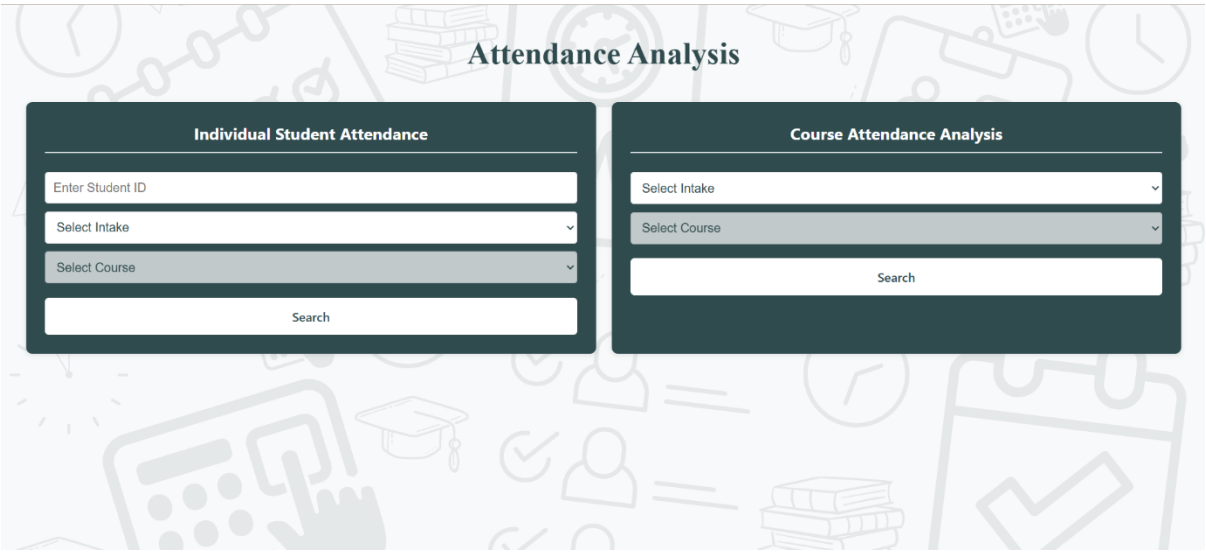
6.4 Attendance Analysis

This AttendanceAnalysisPage React component is designed to display student attendance data either by searching for a specific student or analyzing attendance for an entire intake and course.

Currently, it uses mock data (fake test data) when real backend APIs are not available or not responding correctly. You can search attendance by entering a student ID and selecting their intake and course, or by just selecting an intake and course to see a whole class summary and graph.

It tries to load data using API calls like /api/intakes, /api/courses, and /api/attendance/..., but if those fail, it switches to using default test values. The bar graph is also built from this mock data.

Since the actual API is not returning valid responses yet this page is still under development and will be properly functional in future upgrades once the backend endpoints are correctly connected and returning real data.



7. Simple Future Developments

1. Add More Graphs
    - Show monthly or weekly attendance graphs.
    - Compare subjects or students easily.
    - Download reports as PDF.
2. Make Face Matching Better
    - Use better models to recognize faces.
    - Add more photos with different angles and lighting.
    - Stop fake faces using photos or videos.
3. Emotion Detection
    - Detect if students look happy, sad, or tired.
    - Help find students who may need support.
4. Add Helpful Features
    - Support face recognition with masks.
    - Send alerts for low attendance.
    - Create an admin dashboard to manage everything.

8. Conclusion

This system already makes student attendance easy and automatic using face recognition. In the future, we can add smart graphs, improve accuracy, and even detect emotions to support students' mental well-being. These upgrades will make the system more useful, smart, and caring for both learning and health.